University of Malta
BSc(Hons.) IT Year II

Problem Solving Techniques in Artificial Intelligence

Lecture 2

Search

- Evaluation
- Cutoff
- Alpha Beta Pruning

Department of Computer Science and AI

Assigning Utilities: Evaluation Functions

In chess, evaluation function might be based on

- Material:

pawn	:	1
bishop	:	3
knight	:	3
rook	:	5
queen	:	9

- Position:
  - pawns on final rank
  - check next move
  - fork next move

- Weighted linear evaluation function

$$e = w_1 f_1 + w_2 f_2 +, \ldots, + w_n f_n$$

where f is some feature and w is a weighting that biases the contribution of that feature to the final score.

When to Cut Search

- **Depth Limit:** don't search beyond a given depth.
- **Iterative Deepening:** don't search beyond a given time (i.e. time is up when resources run out).

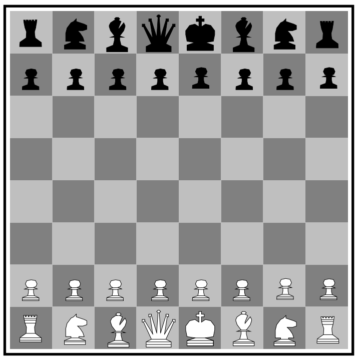
  Assume:
  - 1000 positions/second
  - Branching Factor 35
  - 150 seconds/move
  (tournament play)

  $$N = B^d$$

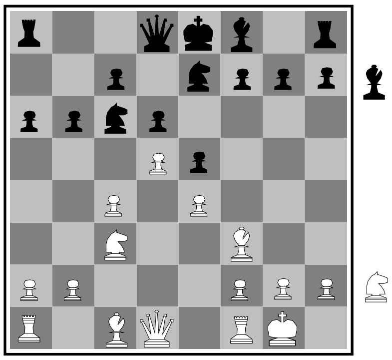
  Max effective ply lookahead = 3-4

- **Pitfalls:**
  - Quiescence: evaluation of non-quiescent positions is likely to yield a false value.
  - Horizon Effect: invisibility of moves "over the horizon" imposed by cutoff

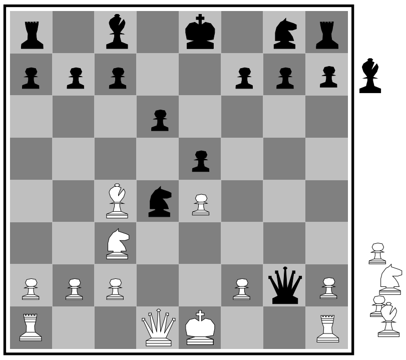
Chess Positions



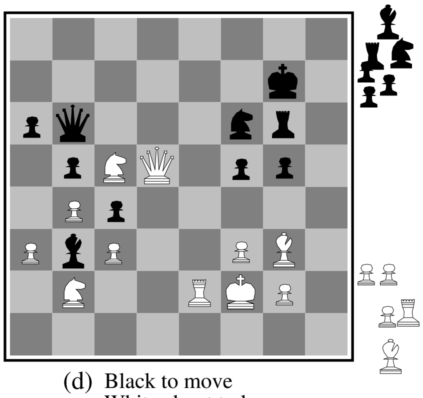
(a) White to move
Fairly even



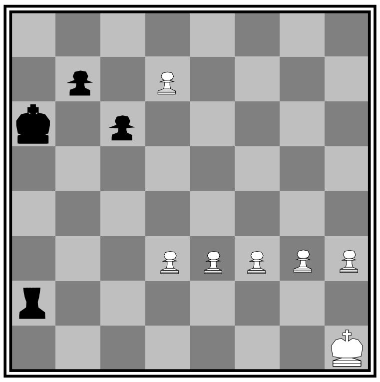
(b) Black to move
White slightly better



(c) White to move
Black winning

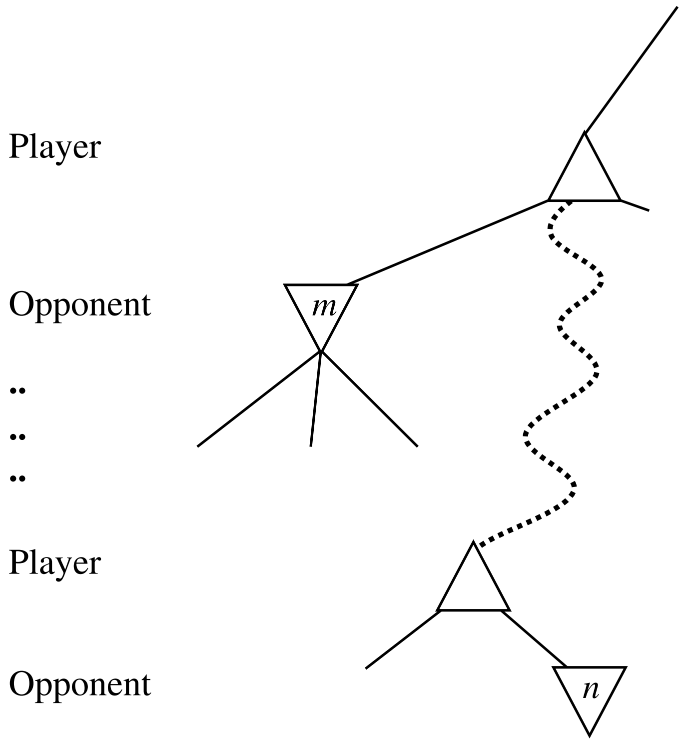


(d) Black to move
White about to lose



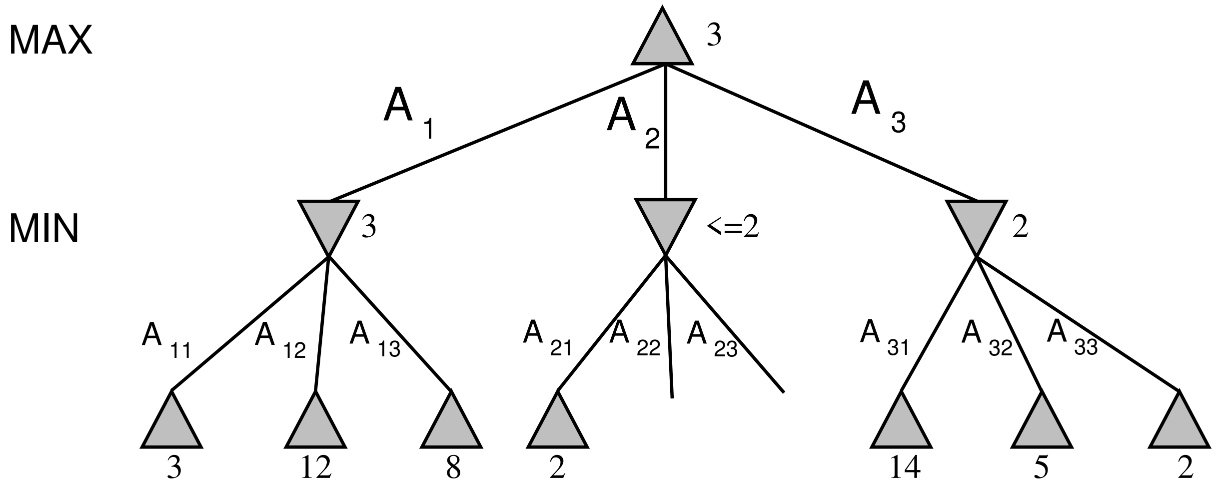
Black to move

Alpha-Beta Pruning General Principle



- Suppose a player has a choice of moving to node n.
- If player has a better choice m either at the parent of n or further up in the tree, n will never be reached.
- Once we have found out enough about n, we can prune it.

Alpha-Beta Pruning



- Initial sequence: $A_{11}12A_{13}A_{21}$ **STOP**
- Clearly, Max will pick the **best** of A_1, A_2, A_3.
- With $A_1 = 3$, $A_2 = 2$ (or less) will never be chosen. Hence, no need to examine further nodes in A_2
- A_3 must still be expanded.

Alpha-Beta Pruning Algorithm

Let α be the value of the best choice so far along the path for Max and β be the lowest value of any choice along the path for Min.

Alpha-beta updates the values for α and β as it goes along, pruning a subtree as soon as it is known to be worse (relatively) than the current α and β value.

function MAX-VALUE(*state, game*, α, β) **returns** the minimax value of *state*
  inputs: *state*, current state in game
        game, game description
        α, the best score for MAX along the path to *state*
        β, the best score for MIN along the path to *state*

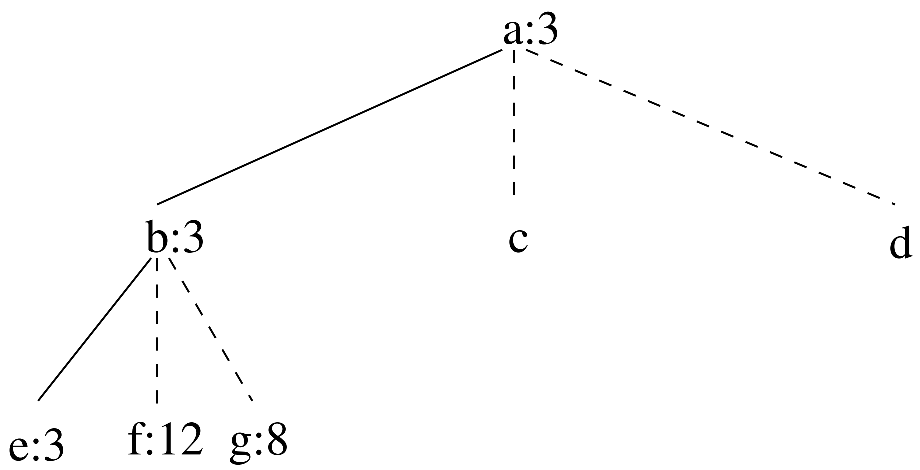
  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each *s* **in** SUCCESSORS(*state*) **do**
    $\alpha \leftarrow$ MAX(α, MIN-VALUE(*s, game*, α, β))
    if $\alpha \geq \beta$ **then return** β
  end
  return α

---

function MIN-VALUE(*state, game*, α, β) **returns** the minimax value of *state*

  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each *s* **in** SUCCESSORS(*state*) **do**
    $\beta \leftarrow$ MIN(β, MAX-VALUE(*s, game*, α, β))
    if $\beta \leq \alpha$ **then return** α
  end
  return β

Alpha Beta Pruning Trace



Initially, $\alpha = -\infty, \beta = +\infty$

maxval(a,α,β)

foreach s in {b,c,d}
  $\alpha \leftarrow$ max(α,minval(s,α,β))
  if $\alpha \geq \beta$ return β

return α

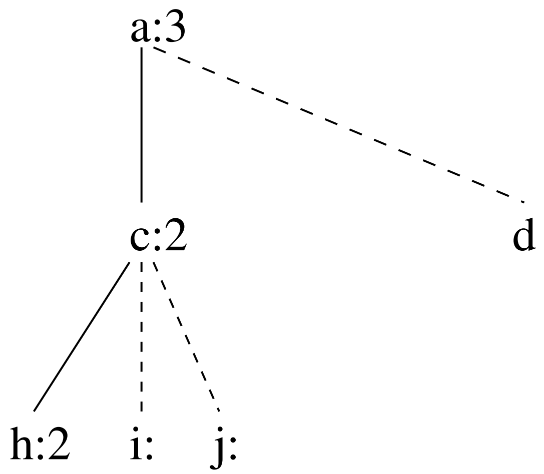
minval(b,α,β)

foreach s in {e,f,g}
  $\beta \leftarrow$ min(β,maxval(s,α,β))
  if $\beta \leq \alpha$ return α

return β

minval(b,α,β) returns 3, so $\alpha = 3$

Alpha Beta Pruning Trace



Now, $\alpha = 3, \beta = +\infty$

maxval(a,α,β)

foreach s in $\{$c,d$\}$ $\alpha \leftarrow$ max(α,minval(s,α,β))
if $\alpha \geq \beta$ return β

return α

minval(c,α,β)

foreach s in $\{$h,i,j$\}$ $\beta \leftarrow$ min(β,maxval(s,α,β))
if $\beta \leq \alpha$ return α

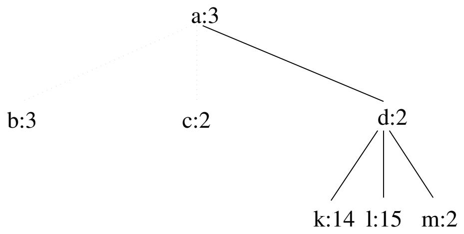
return β

maxval(h,α,β) returns 2, so $\beta = 2$

Because $2 \leq 3$ so minval(c) returns

ignoring maxval(i,α,β), maxval(j,α,β)

Alpha-Beta Pruning Effectiveness



- Node d must still be examined (and in fact will have to be examined in its entirety, since its minval (=2) is not apparent until node n has been evaluated.

- Effectiveness thus depends largely on ordering with which successors are examined.

- With perfect ordering, alpha-beta need only examine O($b^{d/2}$) nodes rather than O(b^d) nodes as with minimax.

- Effective branching factor = $\sqrt{b}$ instead of b, about 6 rather than 35 for chess. Lookahead increases to 8 ply.

Effective Branching Factor

"...If the total number of nodes expanded by A* for a particular problem is N, and the solution depth is d, then the effective branching factor b is the branching factor that a uniform tree of depth d would have to have in order to contain N nodes.."

$$N = 1 + b + b^2 + \ldots + b^d$$

i.e. for known d and N, solve for b. As d grows, N tends towards b^d

$$\begin{array}{lll} \mathrm{N} & \approx & b^d \\ \log(\mathrm{N}) & \approx & \log(b^d) \\ \log(\mathrm{N}) & \approx & \mathrm{d}\ \log(\mathrm{b}) \\ \log(\mathrm{b}) & \approx & \frac{log(N)}{d} \\ \mathrm{b} & \approx & \mathrm{antilog}(\frac{log(N)}{d}) \end{array}$$